Adaptation de la méthode de Davidson à la résolution de systèmes linéaires : implémentation d'une version par blocs sur un multiprocesseur

Miloud Sadkane, Brigitte Vital

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 1240

Programme 7
Calcul Scientifique,
Logiciels Numériques et Ingénierie Assistée

ADAPTATION DE LA METHODE DE DAVIDSON A LA RESOLUTION DE SYSTEMES LINEAIRES : IMPLEMENTATION D'UNE VERSION PAR BLOCS SUR UN MULTIPROCESSEUR

Miloud SADKANE
Brigitte VITAL

Juin 1990

RR_1240

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Adaptation de la méthode de Davidson à la résolution de systèmes linéaires ; implémentation d'une version par blocs sur un multiprocesseur

Miloud SADKANE
CERFACS
42,Av, G. Coriolis, 31057 Toulouse Cedex

Brigitte VITAL
IRISA/INRIA
Campus de Beaulieu
35042 RENNES Cedex

Publication Interne n° 533 - 34 Pages
23 avril 1990

Résumé

La méthode de Davidson est habituellement utilisée dans les problèmes de valeurs propres symétriques. Dans cet article, nous l'adaptons à la résolution de systèmes linéaires creux de grande taille. Les aspects théoriques et pratiques de cette méthode sont étudiés; en particulier nous montrons comment la méthode peut être accélérée à l'aide de préconditionnements. Des essais numériques sont présentés pour lesquels nous avons utilisé une version par blocs de la méthode qui permet la résolution simultanée de plusieurs systèmes de matrice identique et met en évidence des multiplications matrice-matrice ; nous donnons des résultats sur Cray2 [1] qui confirment la bonne efficacité de notre implémentation sur un super calculateur.

Davidson's method for linear systems of equations; implementation of a block algorihtm on a multiprocessor

Abstract

Davidson's method is usually used for symmetric eigenvalue problems. In this paper, it is adapted for the solution of large sparse linear systems of equations. Theoretical and practical aspects are studied; especially it is shown how the method can be accelerated using preconditioning technics. Numerical tests have been performed in which we used a block version of this method which allows simultaneous solution of several systems with the same matrix and allows matrix-matrix multiplications to be performed; we present some results on the Cray2 Computer which exhibit the good efficiency of our implementation on a super computer.

[1] Tous les tests ont été réalisés sur le Cray2 du CCVR à Palaiseau en utilisant le microtasking

1 Introduction

La résolution des systèmes linéaires à matrices symétriques creuses de grande taille définies positives représente un des problèmes fondamentaux dans beaucoup d'applications scientifiques (par exemple les applications utilisant la méthode des éléments finis comme le calcul de structures). Deux types de méthodes sont utilisées: les méthodes dites directes et les méthodes itératives; parmi les méthodes directes citons celles basées sur la factorisation LU, les méthodes frontales au sens de gauss,etc....

Les méthodes itératives s'imposent dès que l'ordre du problème est trop élevé et interdit toute factorisation de la matrice qui dépasserait les limites de la mémoire de la machine; c'est le cas en calcul de structures par exemple où l'on traite actuellement des problèmes à plus de 10^5 degrés de liberté avec une largeur de bande comprise entre 10^2 et 10^4! L'objectif de la catégorie des méthodes itératives dites de sous-espaces, est de construire une suite croissante de sous-espaces V_k desquels on extrait une suite de vecteurs x_k convergeant assez rapidement vers la solution x du problème; le but est que la convergence soit assez rapide pour que les sous-espaces V_k soit de dimension petite par rapport à l'ordre n du problème et qu'ainsi les calculs pour obtenir x_k soient peu coûteux. Les méthodes de Gradients Conjugués ([8] ,[2]) sont les plus connues; citons aussi la méthode appelée GMRES ([11]). Les sous-espaces utilisés V_k sont très souvent des espaces de Krylov ([9] ,[8]) et les différences entre toutes ces méthodes résident dans le critère permettant de calculer le vecteur x_k dans ces sous-espaces (minimisation de l'erreur dans une norme donnée, minimisation du résidu,...); à partir de la même méthode de sous-espaces, on obtient d'autre part des versions différentes en l'accélérant par des préconditionnements différents. Toutes ces méthodes présentent l'avantage de n'utiliser la matrice que dans le noyau *multiplication matrice-vecteur* ce qui permet d'exploiter la structure creuse particulière de cette matrice et d'optimiser la place nécessaire en mémoire.

Dans cet article nous présentons une méthode de sous-espaces, plus précisément nous proposons d'adapter la méthode de Davidson destinée à la recherche des éléments propres extrémaux des matrices symétriques de grande taille, pour résoudre des systèmes linéaires dont la matrice A possède les mêmes caractéristiques; nous nous intéressons de plus à la résolution simultanée de plusieurs systèmes linéaires de matrice identique. l étant un entier inférieur ou égal à n,

il s'agit donc de trouver une matrice rectangulaire X solution du problème:

$$AX = B \ , \ X \in R^{n,l}, B \in R^{n,l}, A \in R^{n,n} \qquad (1)$$

où A est une matrice creuse symétrique inversible d'ordre n élevé et l est "petit" par rapport à n. La résolution de ce problème nous a amené très naturellement à un algorithme par blocs; on connait bien l'avantage des calculs par blocs sur un calculateur vectoriel et parallèle: cela permet d'accroître l'efficacité du calcul en optimisant les accès à la mémoire. Nous verrons d'autre part lors des essais numériques que dans certains cas l'utilisation des blocs accélère la convergence. Pour l'étude théorique nous nous limiterons au cas où A est définie positive bien que l'expérience a montré que l'algorithme peut très bien s'appliquer parfois quand la matrice ne l'est pas (§8.4).

L'idée est ici de remplacer la résolution de (1) par la résolution successive de systèmes $HY = Z$ où H est une matrice symétrique (définie positive) pleine d'ordre assez petit par rapport à celui de A, obtenue par projection sur un certain sous-espace. D'autre part nous utilisons un préconditionnement pour accélérer la convergence; à cette fin nous supposons, dans la suite de cet article, que la matrice A est décomposée sous la forme $A = M - N$ où M est supposée être une matrice symétrique définie positive.

Dans les paragraphes §2,3,4 nous décrivons l'algorithme proposé ainsi que sa mise en oeuvre efficace sur un super calculateur; dans les paragraphes §5,6 nous montrons la convergence de la méthode et étudions sa complexité; les paragraphes §7,8 sont réservés à l'implantation de l'algorithme par blocs sur un calculateur multiprocesseur vectoriel et à des résultats de tests réalisés sur le Cray2 du CCVR et illustrant le bon comportement de notre algorithme sur ce type de machines et le gain en vitesse de convergence que l'on peut obtenir parfois par une résolution simultanée de systèmes, au lieu de résolutions successives, avec cette méthode particulière de sous-espaces.

2 L'algorithme

Soit à résoudre le système (1); l'algorithme de Davidson par blocs s'écrit:

Initialisation : Soit $V_1 \in \mathbf{R}^{n,l}$. $V_1 = MGS(V_1)$ et soit m un entier.

Itération :

pour $k = 1, 2, \ldots$ *faire*

1. $W_k = AV_k$
2. $H_k = V_k^T W_k$; $Z_k = V_k^T B$
3. Résoudre $H_k Y_k = Z_k$
4. $X_k = V_k Y_k$
5. $R_k (= AX_k - B) = (AV_k)Y_k - B$
6. $T_k = M^{-1} R_k$

   - Si $dim(V_k) \leq m$ alors $V_{k+1} = MGS(V_k; T_k)$
   - Sinon $V_{k+1} = MGS(X_k; T_k)$

Remarques:

1) L' entier m est fixé pour limiter le nombre d'itérations internes (i.e : avoir dimension$(V_k) \leq m$, $k = 1, 2, \ldots$) afin de contrôler le volume des calculs et la place mémoire. Si on ne redémarre pas, c.a.d si on choisit $m = n$, la convergence est atteinte en au plus m/l itérations en artithmétique exacte.

2) La notation MGS est réservée à l'algorithme de Gram-Schmidt modifié.

3) Pour le calcul (5) du bloc de résidus R_k, on ne refait pas le produit de V_k par A mais on conserve les produits AV_k obtenus après l'étape (1); il faut donc prévoir une place mémoire de $2mn$ pour stocker les vecteurs V_k et AV_k. Hormis les matrices A et M c'est le seul besoin de place en mémoire significatif, les autres tableaux nécessaires ne dépendant pas de n.

Quelques mots sur les propriétés mathématiques de la méthode:

Le but de l'étape (6) est de corriger les vecteurs colonnes de $X_k = [x_k^1, x_k^2, \ldots, x_k^l]$ avant de les incorporer à la base V_k. Cela peut s'interpréter comme une étape de la méthode du raffinement itératif ([1]), pourvu que $\rho(M^{-1}A) \leq 1$, utilisant les vecteurs x_k^i, $i = 1, 2, \ldots l$ comme solutions approchées et la matrice M comme approximation de A:

$$M(t_k^i - x_k^i) = r_k^i \qquad i = 1, 2, \ldots, l$$

où les t_k^i et r_k^i désignent les colonnes des blocs T_k et R_k définis aux étapes (5) et (6) de l'algorithme (§2).
On peut d'autre part essayer de situer cette méthode par rapport aux méthodes de sous-espaces de référence, en particulier par rapport aux gradients conjugués; plaçons-nous pour cela dans le cas simple (l=1). Remarquons que la résolution du système d'interaction $H_k Y_k = Z_k$ à l'étape (3) est équivalente à la réalisation d'une des deux propriétés suivantes:

1. le résidu est orthogonal au sous-espace E_k engendré par la base V_k:

$$V_k^T R_k = 0.$$

2. X_k réalise le minimum de l'erreur (en norme A) par rapport à la solution exacte X sur le sous-espace E_k: $||X - X_k||_A \leq ||X - V||_A$, $\forall V \in E_k$

D'autre part, en choisissant comme vecteur de départ $V_1 = M^{-1}B$, on vérifie que V_k engendre un espace de Krylov:

$$E_k = span\left(V_1, M^{-1}AV_1, ..., (M^{-1}A)^{k-1}V_1\right)$$

Théoriquement, et si on n'utilise pas de redémarrage, la méthode est donc équivalente à un gradient conjugué préconditionné ou PCG ([2]).

3 Mise en oeuvre de l'algorithme

L'algorithme utilise à chaque itération des produits matrice-vecteur, qu'il convient d'optimiser en exploitant les propriétés de la matrice A. On remarque d'autre part que l'étape $k+1$ de l'algorithme utilise les résultats de l'étape k. En effet, si l'on note $V_{k+1} = [V_k; U_{k+1}]$, où U_{k+1} est une matrice rectangulaire dont les vecteurs colonnes sont ceux calculés à l'étape $k+1$ et qui sont orthogonaux entre eux et à V_k ($dim(U_{k+1}) \leq l$), alors

$$H_{k+1} = \begin{pmatrix} H_k & B_k \\ B_k^T & C_k \end{pmatrix} \tag{2}$$

$$Z_{k+1} = \begin{pmatrix} Z_k \\ D_k \end{pmatrix} \tag{3}$$

Avec $B_k = V_k^T A U_{k+1}$, $C_k = U_{k+1}^T A U_{k+1}$ et $D_k = U_{k+1}^T B$

L'algorithme doit donc prendre en compte le fait que les matrices H_{k+1} ($resp.$ Z_{k+1}) s'obtiennent à partir de H_k ($resp.$ Z_k) en ajoutant les colonnes $\begin{pmatrix} V_k^T A U_{k+1} \\ U_{k+1}^T A U_{k+1} \end{pmatrix}$

(resp. les lignes $U_{k+1}^T B$). Compte tenu de la symétrie de H_{k+1}, les lignes et les colonnes que l'on rajoute sont identiques. La résolution des systèmes linéaires $H_k Y_k = Z_k$ à l'étape (3) s'effectue par exemple par la méthode de Cholesky ou plutôt par une décomposition LDL^T (§4). La convergence de l'algorithme est mesurée par la norme du résidu $\| R_k \|$ multiplié par un certain coefficient (§8), le choix de la norme n'a pas d'importance. La matrice V_{k+1} est obtenue à partir de V_k en rajoutant les vecteurs colonnes de la matrice $T_k = M^{-1} R_k$ après les avoir orthonormés entre eux et par rapport à V_k. Lorsque la dimension de V_k devient supérieure à m, et si l'algorithme n'a pas convergé, on redémarre avec une base orhonormée du sous espace engendré par $[X_m, T_m]$.

4 Résolution du système d'interaction: $H_k Y_k = Z_k$

Les matrices $H_k, k = 1, ..., m$ étant définies positives, on peut procéder par factorisation de Cholesky; d'un point de vue pratique, il est plus intéressant d'utiliser la factorisation LDL^T, où D est une matrice diagonale, L est une matrice triangulaire inférieure de diagonale unité, qui, contrairement à la décomposition de Cholesky, ne nécessite pas d'extraction de racines carrées. Cette décomposition existe et est unique pour toute matrice symétrique inversible, mais une matrice symétrique définie positive se caractérise par le fait que les éléments de D sont strictement positifs. En fait, il suffit dans notre situation, de mettre à jour la factorisation de la matrice H_{k+1} à partir de celle de la matrice précédente H_k, en utilisant la propriété suivante:

Proposition 1:

Si $\bar{A} = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix}$ où: $A \in R^{n,n}, C \in R^{p,p}, B \in R^{n,p}$,

si les matrices A et $\bar{A}$ sont définies positives et si $A = LDL^T$ désigne la factorisation de A, alors la factorisation de $\bar{A}$ est: $\bar{A} = \bar{L}\bar{D}\bar{L}^T$ avec:

$$\bar{L} = \begin{pmatrix} L & 0 \\ K^T & l \end{pmatrix} \quad \text{et} \quad \bar{D} = \begin{pmatrix} D & 0 \\ 0 & d \end{pmatrix}$$

où les matrices $K \in R^{n,p}$, l et $d \in R^{p,p}$ sont solutions des systèmes:

$$(LD)K = B \tag{4}$$

$$ldl^T = C - K^T D K \tag{5}$$

Preuve:

L'équation (4) donne $K = (LD)^{-1} B$; donc l'équation (5) a une solution si et

seulement la matrice $C - K^T DK = C - B^T A^{-1} B$ est définie positive; ce qui est le cas car:

$$\bar{A} = \begin{pmatrix} I_n & 0 \\ B^T A^{-1} & I_p \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & C - B^T A^{-1} B \end{pmatrix} \begin{pmatrix} I_n & A^{-1} B \\ 0 & I_p \end{pmatrix} \qquad (6)$$

donc les deux matrices $\bar{A}$ et $\begin{pmatrix} A & 0 \\ 0 & C - B^T A^{-1} B \end{pmatrix}$ ont même inertie. □

L'équation (4) consiste en plusieurs systèmes linéaires à matrices triangulaires inférieures inversibles. l'équation (5) est une factorisation LDL^T d'une matrice plus petite. Cette propriété peut être utilisée pour factoriser complètement la matrice en réitérant le procédé, ce qui permet d'introduire des calculs par blocs (multiplications matrice-matrice) et donc de gagner en efficacité sur un calculateur vectoriel ([8]).
Ici cette méthode est utilisée pour obtenir la factorisation $L_{k+1} D_{k+1} L_{k+1}^T$ de la matrice définie positive H_{k+1} à partir de celle de H_k qui est aussi définie positive; on diminue ainsi notablement le coût des résolutions successives des systèmes d'interaction (voir la remarque 2 qui suit). notons que, d'après la preuve de la proposition 1, la matrice:

$$C_k - B_k^T H_k^{-1} B_k$$

est définie positive.

Remarques.

1. La résolution de $H_k Y_k = Z_k$ avec Y_k, Z_k dans $R^{k,l}$ est alors remplacée par la résolution de deux systèmes triangulaires :

$$\begin{cases} L_k X = Z_k \\ D_k L_k^T Y_k = X \end{cases}$$

   dont la complexité est $\left(\frac{kl(kl-1)}{2} + \frac{kl(kl-1)}{2} + kl\right) l = ((kl)^2) l$ puisqe L_k et D_k sont des matrices d'ordre kl.
   (Nous ne tenons compte dans tous nos calculs de complexité, que des multiplications et divisions flottantes).

2. Sachant que la complexité de la factorisation LDL^T d'une matrice d'ordre n est de $\frac{n^3}{6}$, on en déduit la complexité totale : $sol(k)$ de l'étape de

résolution par cette méthode :

$$\underbrace{\underbrace{[\frac{((k-1)l)^2+(k-1)l}{2}]l}_{\text{résolution de (4)}} + \underbrace{(l^2+l)(k-1)l + \frac{l^3}{6}}_{\text{résolution de (5)}}}_{\text{mise à jour factorisation } LDL^T} + \underbrace{(kl)^2 l}_{\text{résolution}}$$

Remarquons que si on avait décomposé directement H_k sous la forme LDL^T on aurait obtenu une complexité de : $\frac{(kl)^3}{6} + (kl)^2 l$; pour k assez grand la méthode proposée est donc très avantageuse.

3. L'algorithme peut s'appliquer à des matrices non symétriques. Dans ce cas la complexité est relativement plus élevée puisqu'il faut à chaque itération calculer deux produits matrice-vecteur du type $u = Av$; $u = A^T v$. La résolution du système linéaire interne $H_k Y_k = B_k$ peut s'effectuer par la méthode de Gauss en décomposant la matrice H_k sous la forme $H_k = L_k U_k$ avec L_k triangulaire inférieure à diagonale unité et U_k une matrice triangulaire supérieure. Cette décomposition est utilisée à l'étape k de l'algorithme tout en profitant de l'étape antérieure par un procédé analogue à celui donné à la proposition 1. Il faut noter qu'en général, cette décomposition peut échouer.

5 Complexité de l'algorithme davsys.

Dans ce paragraphe nous comptons le nombre de multiplications (et divisions) présentes dans l'algorithme. Considérons le cas ponctuel (i.e : $l = 1$), le calcul se généralisant facilement au cas par blocs.

Notons N_a la complexité du produit matrice-vecteur. D'après les formules (2), (3) seule la dernière colonne de H_k (resp. la dernière ligne de Z_k) est à construire et elle nécessite kn (resp. n) multiplications. Nous désignons par $Sol(k)$ (resp. N_r) la complexité de l'étape qui consiste à résoudre le système linéaire $H_k Y_k = Z_k$ (resp. l'étape $T_k = M^{-1} R_k$).

Sans redémarrage, on effectue :

$$\sum_{k=1}^{m} (N_a + 4kn + 4n + N_r + Sol(k)) \text{ multiplications.}$$

D'autre part, d'après la remarque 2 du paragraphe §3 et enprenant $l = 1$ on a :

$$Sol(k) = \frac{3k^2}{2} + \frac{3k}{2} - \frac{11}{6}$$

D'où une complexité de :

$$m\left[N_a + 2n(m+1) + 4n + N_r + \frac{1}{4}(m+1)(2m+1) - \frac{11}{6} + \frac{3}{4}(m+1)\right].$$

Le coût de cette méthode est dominée en particulier par N_a la complexité du produit matrice-vecteur, et par N_r la complexité du préconditionnement ; ces deux quantités sont de l'ordre de $O(n)$ pour les rangements creux. D'autre part, les coûts de l'orthogonalisation, de la mise à jour de la matrice H_k à partir de H_{k-1}, du calcul de $X^k = V^k Y^k$ et du résidu R_k sont aussi importants . Une bonne mise en oeuvre de ces étapes est donc nécessaire.

6 Résultat de convergence.

Théorème 1 . Si la matrice A est symétrique définie positive, la suite des itérés X_k converge vers la solution $X = A^{-1}B$ et le facteur de convergence est majoré par $\frac{cond_2(M^{-1}A)-1}{cond_2(M^{-1}A)+1}$.

$cond_2(M^{-1}A)$ désigne le conditionnement de la matrice $M^{-1}A$ relativement à la norme euclidienne.

Démonstration. Nous supposons de plus que les vecteurs colonnes des trois blocs R_k, $M^{-1}R_k$ et $(I - V_k V_k^T)M^{-1}R_k$ (qui est la projection orthogonale de $M^{-1}R_k$ sur l'orthogonal de V_k) sont linéairements indépendants [2] à chaque étape k avant la convergence .

Il est clair que les matrices H_k, $k = 1, 2, \ldots$ sont définies positives .

Soit $H_{k+1}^{-1} = \begin{pmatrix} E_k & F_k \\ F_k^T & G_k \end{pmatrix}$ l'inverse de la matrice H_{k+1}. Un calcul élémentaire donne :

$$G_k = (C_k - B_k^T H_k^{-1} B_k)^{-1} \tag{7}$$

$$E_k = H_k^{-1}(I + B_k G_k B_k^T H_k^{-1}) \tag{8}$$

$$F_k = -H_k^{-1} B_k G_k \tag{9}$$

Et par suite :

$$Y_{k+1} = H_{k+1}^{-1} Z_{k+1} = \begin{pmatrix} E_k Z_k + F_k D_k \\ F_k^T Z_k + G_k D_k \end{pmatrix} = \begin{pmatrix} Y_k + H_k^{-1} B_k G_k (B_k^T Y_k - D_k) \\ -G_k (B_k^T Y_k - D_k) \end{pmatrix}$$

En remplaçant D_k par sa valeur $U_{k+1}^T B$, on en déduit

[2] Ceci est licite puisque l'utilisation de MGS permet, le cas échéant, d'éliminer les colonnes qui seraient liées

$$Y_{k+1} = \begin{pmatrix} Y_k \\ 0 \end{pmatrix} + \begin{pmatrix} H_k^{-1} B_k G_k U_{k+1}^T R_k \\ -G_k U_{k+1}^T R_k \end{pmatrix} \tag{10}$$

De même

$$X_{k+1} = X_k + (V_k H_k^{-1} V_k^T A - I) U_{k+1} G_k U_{k+1}^T R_k \tag{11}$$

$$R_{k+1} = R_k + A(V_k H_k^{-1} V_k^T A - I) U_{k+1} G_k U_{k+1}^T R_k \tag{12}$$

Dans la suite, nous utilisons la norme matricielle suivante :
Si $U \in \mathbf{R}^{n,p} \quad n, p \in \mathbf{N} \quad n \geq p \quad \| U \|_A = (Trace(U^T A U))^{\frac{1}{2}}$.
Soit X la solution du problème (1), alors

$$\| X_{k+1} - X \|_A^2 = Trace((X_{k+1} - X)^T A (X_{k+1} - X))$$
$$= Trace((X_k - X)^T A (X_k - X) - R_k^T U_{k+1} G_k U_{k+1}^T R_k)$$

D'où

$$\| X_{k+1} - X \|_A^2 = \| X_k - X \|_A^2 (1 - \frac{Trace(R_k^T U_{k+1} G_k U_{k+1}^T R_k)}{Trace(R_k^T A^{-1} R_k)}) \tag{13}$$

Remarquons que puisque la matrice A^{-1} est symétrique définie positive et que les colonnes de R_k sont linéairement indépendantes, il en résulte que la matrice symétrique $R_k^T A^{-1} R_k$ est aussi définie positive, en particulier elle est inversible.

Nous allons simplifier l'expression ci-dessus en explicitant U_{k+1}. D'après l'algorithme, U_{k+1} est une matrice dont les vecteurs colonnes sont orthogonaux entre eux et à V_k. Si $Q_k \Phi_k$ désigne la décomposition QR de la matrice $(I - V_k V_k^T) M^{-1} R_k$, il en résulte que $U_{k+1} \equiv Q_k$ et d'autre part puisque les vecteurs colonnes de U_{k+1} sont indépendants, la matrice Φ_k est inversible. Soit $U_{k+1} = (I - V_k V_k^T) M^{-1} R_k \Phi_k^{-1}$. Donc

$$\begin{aligned} G_k^{-1} &= C_k - B_k^T H_k^{-1} B_k \\ &= U_{k+1}^T A U_{k+1} - U_{k+1}^T A V_k H_k^{-1} V_k^T A U_{k+1} \\ &= \Phi_k^{-T} R_k^T M^{-1} (A - A V_k H_k^{-1} V_k^T A) M^{-1} R_k \Phi_k^{-1} \end{aligned}$$

On a déja remarqué (§4) que la matrice symétrique $C_k - B_k^T H_k^{-1} B_k$ est définie positive, ce qui entraîne que la matrice symétrique $R_k^T M^{-1} (A - A V_k H_k^{-1} V_k^T A) M^{-1} R_k$ est aussi définie positive. Posons

$$J_k = A - A V_k H_k^{-1} V_k^T A.$$

Alors,

$$\| X_{k+1} - X \|_A^2 = \| X_k - X \|_A^2 (1 - \eta_k)$$

avec

$$\eta_k = \frac{Trace(R_k^T M^{-1} R_k (R_k^T M^{-1} J_k M^{-1} R_k)^{-1} R_k^T M^{-1} R_k)}{Trace(R_k^T A^{-1} R_k)}$$

On a d'autre part :

$$\begin{aligned}
&Trace(R_k^T M^{-1} R_k (R_k^T M^{-1} J_k M^{-1} R_k)^{-1} R_k^T M^{-1} R_k) \geq \\
&\rho(R_k^T M^{-1} R_k (R_k^T M^{-1} J_k M^{-1} R_k)^{-1} R_k^T M^{-1} R_k) = \\
&\rho([(R_k^T M^{-1} R_k)^{-1} (R_k^T M^{-1} J_k M^{-1} R_k)(R_k^T M^{-1} R_k)^{-1}]^{-1}) = \\
&\frac{1}{\lambda_{min}[(R_k^T M^{-1} R_k)^{-1} (R_k^T M^{-1} J_k M^{-1} R_k)(R_k^T M^{-1} R_k)^{-1}]} \geq \\
&\frac{1}{\lambda_{min}[(R_k^T M^{-1} R_k)^{-1} (R_k^T M^{-1} A M^{-1} R_k)(R_k^T M^{-1} R_k)^{-1}]} \geq \\
&\| R_k^T M^{-1} R_k (R_k^T M^{-1} A M^{-1} R_k)^{-1} R_k^T M^{-1} R_k \|_2 .
\end{aligned}$$

Donc

$$\| X_{k+1} - X \|_A^2 \leq \| X_k - X \|_A^2 (1 - \frac{\| R_k^T M^{-1} R_k (R_k^T M^{-1} A M^{-1} R_k)^{-1} R_k^T M^{-1} R_k \|_2}{\| R_k^T A^{-1} R_k \|_2}).$$

Avec

$$\begin{aligned}
\| R_k^T A^{-1} R_k \|_2 &= \rho(R_k^T A^{-1} R_k) \\
&= \rho[(R_k^T M^{-1} R_k)^{-1} (R_k^T A^{-1} R_k)(R_k^T M^{-1} R_k)]
\end{aligned}$$

$$\begin{aligned}
= \rho[&(R_k^T M^{-1} R_k)^{-1} (R_k^T A^{-1} R_k)(R_k^T M^{-1} A M^{-1} R_k)\ (R_k^T M^{-1} R_k)^{-1} \\
&(R_k^T M^{-1} R_k)(R_k^T M^{-1} A M^{-1} R_k)^{-1} (R_k^T M^{-1} R_k)]
\end{aligned}$$

$$\begin{aligned}
\leq \| &(R_k^T M^{-1} R_k)^{-1} (R_k^T A^{-1} R_k)(R_k^T M^{-1} A M^{-1} R_k)\ (R_k^T M^{-1} R_k)^{-1} \|_2 \\
&\| (R_k^T M^{-1} R_k)(R_k^T M^{-1} A M^{-1} R_k)^{-1} R_k^T M^{-1} R_k) \|_2
\end{aligned}$$

D'où

$$\| X_{k+1} - X \|_A^2 \leq \| X_k - X \|_A^2 (1 - \xi_k)$$

Avec

$$\xi_k = \frac{1}{\| (R_k^T M^{-1} R_k)^{-1} (R_k^T A^{-1} R_k)(R_k^T M^{-1} A M^{-1} R_k)(R_k^T M^{-1} R_k)^{-1} \|_2}$$

Nous utilisons à présent l'inégalité de Kantorovic généralisée [3] :

$$\| (R_k^T M^{-1} R_k)^{-1} (R_k^T A^{-1} R_k) (R_k^T M^{-1} A M^{-1} R_k) (R_k^T M^{-1} R_k)^{-1} \|_2 \leq \frac{(Cond_2(M^{\frac{-1}{2}} A M^{\frac{-1}{2}}) + 1)^2}{4\ Cond_2(M^{\frac{-1}{2}} A M^{\frac{-1}{2}})}$$

D'où

$$\| X_{k+1} - X \|_A^2 \leq \| X_k - X \|_A^2 \left(\frac{cond_2(M^{-1}A) - 1}{cond_2(M^{-1}A) + 1}\right)^2$$

Et finalement

$$\| X_k - X \|_A \leq \| X_1 - X \|_A \left(\frac{cond_2(M^{-1}A) - 1}{cond_2(M^{-1}A) + 1}\right)^{k-1}$$ □

7 Implantation sur un multiprocesseur

Le calcul de complexité (§5) montre que le choix du produit matrice-vecteur et du préconditionnement est fondamental dans le comportement de l'algorithme; de plus, sur un multiprocesseur, l'efficacité dependra de la qualité de la parallélisation de ces noyaux. Nous présentons maintenant les diverses solutions que nous avons utilisées.

7.1 Produit Matrice-Bloc de vecteurs

Il s'agit de calculer le produit : $Y = A * X \quad (P)$
où X et Y sont dans $R^{n,k}$, et A est une matrice creuse symétrique d'ordre n.
Le choix de la méthode est lié au type de stockage utilisé pour la matrice.

7.1.1 Produit 'creux'

Dans le stockage 'creux' par lignes, souvent appelé "A-IA-JA", seuls les éléments non nuls de la matrice sont stockés ([7]) . Il existe deux algorithmes, pour calculer le produit (P) suivant que l'on tient compte de la symétrie en stockant seulement la partie triangulaire inférieure L de la matrice A ou pas. :
Cas non symétrique :

$algorithme(1)$

```
do 10 j = 1,k
  Y[1:n,j] = 0
  do 20 i = 1,n
    do 30 l = IA(i),IA(i)+d_i-1
    30  Y(i,j) = Y(i,j)+A(l)*X(JA(l),j)
  20 continue
10 continue
```

Cas symétrique :

$algorithme(2)$

```
do 10 j = 1,k
  Y[1:n,j] = D[1:n]*X[1:n,j]
  do 20 i = 1,n
    do 30 l = IA(i),IA(i)+d_i-2
       Y(i,j) = Y(i,j)+A(l)*X(JA(l),j)
    30  Y(JA(l),j) = Y(JA(l),j)+A(l)*X(i,j)
  20 continue
10 continue
```

où le tableau $D[1:n]$ contient la diagonale de la matrice et d_i est le degré de la ligne i (nombre d'éléments non nuls). Le stockage creux est celui qui demande

le moins de place mémoire, mais les algorithmes associés ne sont pas très efficaces sur un processeur vectoriel : en effet, les degrés des lignes : $(d_i\ ,\ i = 1, n)$ ne sont pas assez importants en général pour permettre d'atteindre une vitesse satisfaisante dans les instructions vectorielles ([7]).
Il y a deux possiblités pour utiliser le parallélisme qui conduisent à deux produits creux différents:

1. *répartir les k colonnes de X et Y sur les processeurs.* On peut alors ne stocker que la partie triangulaire inférieure de la matrice ; cela revient à utiliser l'algorithme (1) en exécutant en parallèle la boucle j ; la boucle {i} est séquentielle et la boucle {l} est vectorielle ; nous appelons ce produit: *produit 1.* Il a l'avantage d'être associé à un stockage optimum pour une matrice symétrique ; mais il ne convient pas au cas $k = 1$ d'un seul vecteur.

2. *répartir la matrice elle-même sur les processeurs.* La répartition par lignes nous amène à stocker entièrement A, sans tenir compte de la symétrie. Si les données se mettent sous la forme :

$$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_{nproc} \end{bmatrix}, X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_{nproc} \end{bmatrix}, Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_{nproc} \end{bmatrix}$$

où *nproc* est le nombre de processeurs, le produit (P) s'écrit :

$$\begin{array}{ll} do\ 10\ ip = 1, nproc & \\ 10 & Y_{ip} = A_{ip} * X_{ip} \end{array}$$

chacun des sous-produits : $A_{ip} * X_{ip}$ se calculant par l'algorithme (1) et la boucle {ip} étant exécutée en parallèle. nous appelons ce produit : *produit 2.* Ici l'efficacité est toujours maximum quelle que soit la valeur de k mais le stockage est deux fois plus volumineux.

7.1.2 Produit bande

Pour certaines matrices, comme celles issues des éléments finis, il existe un stockage plus adapté, dit *stockage bande*, qui consiste à stocker la matrice par diagonales et à ne garder que celles qui comportent des éléments non nuls ; un tableau rectangulaire de réels : $A[1 : n; 0 : nb]$ contient, dans ses $nb + 1$ colonnes les $nb + 1$ diagonales, qui occupent les dernières positions des colonnes ; un tableau d'entiers : $ipt[0 : nb]$ donne la position réelle des diagonales dans la

matrice : le numéro de la diagonale stockée dans la colonne l est $ipt(l)$ (c.a.d. que la différence $|i-j|$ entre les indices de lignes et de colonnes de ses eléments est : $ipt(l)$). Ici nous ne stockons que la partie triangulaire inférieure et nous supposons que la première colonne de A contient la diagonale principale ; le produit associé s'écrit :

```
                do 10 j = 1, k
                  Y[1 : n, j] = A[1 : n, 0] * X[1 : n, j]
                  do 20 l = 1, nb
                    p = ipt(l)
(3)                   do 30 i = 1, n - p
                        Y(i, j) = Y(i, j) + A(i + p, l) * X(i + p, j)
                      30 Y(i + p, j) = Y(i + p, j) + A(i + p, l) * X(i, j)
                  20 continue
                10 continue
```

Remarquons que la longueur des vecteurs est importante (de l'ordre de n) si les diagonales stockées ne sont pas trop excentrées ; c'est par exemple le cas des matrices issues des éléments finis obtenues par discrétisation sur une grille. D'autre part il n'y a pas d'indirections dans les instructions vectorielles ; ce produit est par conséquent très efficace sur un processeur vectoriel quand le stockage bande n'est pas trop volumineux par rapport au stockage creux ([7]). Ce produit peut se paralléliser en utilisant la répartition des colonnes de X et Y sur les processeurs, c.a.d en traitant en parallèle la boucle {j} , les deux boucles {i} étant vectorielles ; on peut aussi utiliser une répartition de chaque diagonale sur les processeurs, par exemple en remplaçant la boucle {i} par une boucle externe de *nproc* itérations qui sera exécutée en parallèle, et une boucle interne de $(n-p)/nproc$ itérations qui sera vectorisée; nous appelerons ces deux produits bande respectivement *produit 3* et *produit 4*.

7.2 Préconditionnement

Il s'agit de calculer l'expression : $Y = M^{-1} * X$, $(Prec.)$
où X et Y sont dans $R^{n,k}$, et M est une matrice creuse symétrique d'ordre n. Nous avons utilisé des préconditionnements différents ; les deux plus simples sont $M = D$ où D est la diagonale de A et $M = T$ où T est la matrice tridiagonale symétrique composée des trois diagonales centrales de A. Pour certains problèmes ces deux choix se sont avérés insuffisants, aussi avons-nous utilisé des préconditionnements dits **ILU** (Incomplet LU Decomposition) : ILU0 et ILUTH ([10]).

7.2.1 Préconditionnement diagonal

Pour la parallélisation de l'instruction (*Prec.*), nous pouvons ici aussi bien utiliser la répartion des colonnes de X et Y aux processeurs que la répartition de la matrice diagonale D elle-même :

$$D = diag(D_1, D_2, \ldots, D_{nproc})$$

Le choix sera systématiquement le même que pour le produit utilisé.

7.2.2 Préconditionnement tridiagonal

La matrice T est factorisée une fois pour toute sous la forme : LDL^t avec L bi-diagonale, le calcul (*Prec.*) consiste alors à résoudre les deux systèmes triangulaires :

$$LD * Z = X \ , \ L^tY = Z$$

La résolution d'un système triangulaire avec un seul second membre étant difficilement parallélisable, nous n'avons implémenté ce noyau qu'avec répartition des colonnes de X et Y aux processeurs. Autrement dit, nous résolvons en parallèle des systèmes linéaires avec second membre unique.

7.2.3 Préconditionnements ILU0 et ILUTH

Les préconditionnements de type ILU consistent à utiliser une factorisation incomplète : $M = L'U'$ de A ; le calcul : $Y = M^{-1}X$ se décompose en deux résolutions de systèmes linéaires :

$$\begin{aligned} L'Z &= X \\ U'Y &= Z \end{aligned}$$

Il y a différentes versions suivant la façon dont est interprétée le terme *"incomplète"* :

1. La version **ILU0** consiste à négliger le remplissage au cours de la factorisation, c.a.d que les matrices L' et U' auront des éléments nuls là où A en avait déjà.
2. La version **ILUTH** consiste à remplacer par 0 les éléments inférieurs en valeur absolue à un certain seuil *threshold* à définir.

De la même façon que pour $M = T$, ces deux préconditionnements sont parallélisés par répartition du second membre aux processeurs uniquement.

Le comportement de l'algorithme est étroitement lié au coefficient $cond(M^{-1}A)$ mis en évidence plus tôt; on s'attend donc à ce que le choix de préconditionnement $M = T$ soit plus efficace que le choix $M = D$ au sens où il conduit à moins d'itérations pour atteindre la convergence, puisque T "approche" mieux A que ne le fait D. Cependant le préconditionnement tridiagonal est évidemment plus coûteux ($3n$ flop) que le diagonal(n flop), et il est plus difficilement parallélisable . Quand aux préconditionement ILU, le coût dépend du nombre nz' d'éléments non nuls de la matrice L' de la décomposition LU incomplète; pour le cas ILU0, nz' est égal au nombre d'éléments non nuls de la matrice nz; mais dans le cas ILUTH, il y a un remplissage au cours de la factorisation et nz' peut être très supérieur à nz.

Outre le choix du produit matrice-vecteur et du préconditionnement, celui des vecteurs de départ V_1 est important; nous avons décidé de prendre systématiquement

$$V_1 = M^{-1}B$$

où M est le préconditionnement utilisé et B le second membre du système linéaire.

7.3 Réorthogonalisation

Un autre point coûteux de l'algorithme, qu'il faut donc essayer de réaliser en parallèle, est la réorthogonalisation; il s'agit d'implémenter l'étape :

$$V = MGS(X,T)$$

avec V dans $R^{n,p+l}$, T dans $R^{n,l}$, X dans $R^{n,p}$, $p \geq l$ et les colonnes de X étant déjà orthonormées; l'algorithme s'écrit :

algorithme(MGS2)

Phase 1 :
do 10 $j = 1, l$
$v_{p+j} = (I - x_p x_p^T)...(I - x_2 x_2^T)(I - x_1 x_1^T)t_j$
10 *continue*
Phase 2 (*algorithme MGS pour l vecteurs*) :
do 30 $j = 1, l$
$v_{p+j} = v_{p+j}/||v_{p+j}||$
do 40 $i = j + 1, l$
$v_{p+i} = v_{p+i} - (v_{p+i}^t v_{p+j})\ v_{p+j}$
40 *continue*
30 *continue*

où la notation u_i désigne pour une matrice U le $i^{ème}$ vecteur colonne de U. La complexité de cette étape est de :

$$2lpn\ (phase\ 1)\ +\ (l^2n + ln)\ (phase\ 2)$$

soit : $ln(l + 2p + 1)$.

La parallélisation de la la *phase* 1 est immédiate: la boucle {j} est traitée en parallèle puisque ses itérations sont indépendantes; dans la *phase* 2 la boucle {i} seule est traitée en parallèle, la boucle extérieure {j} étant nécesssairement séquentielle.

8 Résultats

Les tests ont été réalisés sur le Cray2 du CCVR, qui a quatre processeurs vectoriels, et en utilisant le microtasking; le test d'arrêt est le suivant :

$$||R_k|| \ < \ \epsilon||B||$$

où la notation $||X||$ pour X dans $R^{n,k}$ signifie :

$$sup\,\{||X_j||_2\ ,\ j = 1, \ldots, k\}$$

où les X_j désignent les colonnes de X.

Pour un seul second membre : b, cela revient à estimer l'erreur relative par rapport à la solution x dans la norme définie par la matrice symétrique définie positive : $M = A^tA$:

$$\frac{||R_k||_2}{||b||_2} \ = \ \frac{||x - x_k||_M}{||x||_M}$$

([2]). Nous notons $NBMAX$ la taille maximum de la base V autorisée; rappelons que quand cette taille est atteinte il y a un "redémarrage", c.a.d que l'on commence une nouvelle itération externe (mais cette fois avec comme vecteurs de départ les derniers vecteurs de Ritz obtenus et les résidus associés après correction par M^{-1}); d'autre part nous limitons le nombre de redémarrages à la valeur : $ITMAX$ et nous dirons que l'algorithme n'a pas convergé si cette valeur est atteinte sans que le test d'arrêt soit satisfait.

Nous avons aussi choisi de redémarrer dans le cas où une des k solutions est trouvée; dans ce cas nous éliminons le système linéaire à un second membre associé et nous redémarrons de la même manière que dans le cas où la base a atteint la taille limite. Dans tout ce qui suit nous nous interessons en particulier à deux aspects du comportement de l'algorithme:

1. le nombre d'itérations pour atteindre la convergence qui est aussi le nombre de produits $Y \ = \ A * X$ effectués; en fait, nous donnons plutôt le nombre de multiplications de la matrice A par *un* vecteur noté : **nmult**, car le nombre k de vecteurs ajoutés à la base n'est pas toujours le même;

2. le temps d'exécution et l'accélération obtenue.

8.1 Exemples 1 et 2 (figures 1 et 2 , tableaux 1 et 2)

Nous utilisons ici le format 'creux' et le produit 1 associé (puisqu'il nécessite deux fois plus de place en mémoire pour la matrice, le produit 2 s'impose uniquement dans le cas $k < nproc$); nous engendrons des matrices aléatoires d'ordre n, de densité donnée xp (la densité est définie par $xp = 2nz/n(n+1)$). La première matrice (matrice 1) que nous définissons est ainsi construite:

- les éléments diagonaux sont tirés aléatoirement dans l'intervalle $[n/2+1, n/2+1+diagscal]$
- les éléments hors-diagonaux sont tirés aléatoirement dans l'intervalle $[-subscal/2, +subscal/2]$

avec: $n = 2000$, $xp = 0.16$, $diagscal = 200$ et $subscal = 20$. Nous obtenons par ce procédé une matrice *diagonalement* dominante ayant $nz = 321819$ éléments non nuls dans sa partie triangulaire inférieure.
La seconde matrice (matrice 2) est construite différemment:

- les éléments diagonaux sont tirés aléatoirement dans l'intervalle $[n/2+1+diagscal, n/2+1+2diagscal]$
- les éléments de la sous-diagonale sont tirés aléatoirement dans l'intervalle $[0, diagscal]$
- les autres éléments sont tirés aléatoirement dans l'intervalle $[-subscal/2, +subscal/2]$

avec: $n = 1000$, $xp = 0.3$, $diagscal = 100$ et $subscal = 0.1$. nous obtenons par ce procédé une matrice *tridiagonalement* dominante ayant $nz = 151728$ éléments non nuls dans sa partie triangulaire inférieure. Dans les deux cas, le second membre B est tiré aléatoirement dans [-1,1]. Les figures 1 et 2 illustrent le comportement de l'algorithme pour ces deux matrices quand on résout un système linéaire à $k = 4$ seconds membres avec préconditionnement diagonal ou tridiagonal ou pas de préconditionnement du tout.
Remarque : nous portons en ordonnée la moyenne des expressions $-log_{10}(||r_j||_2)$ pour $j = 1, ..., k$ où r_j est le résidu associé au j^{eme} vecteur: $r_j = A * x_j - b_j$

Evolution du résidu au cours des itérations:
($k = 4, NBMAX = 24, \epsilon = 10^{-10}$,produit 1).

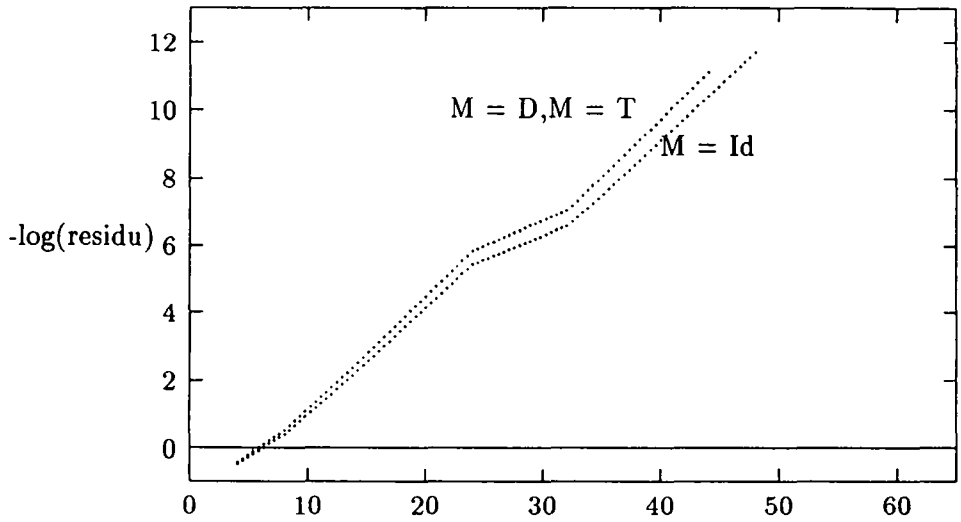

figure 1 (matrice 1)

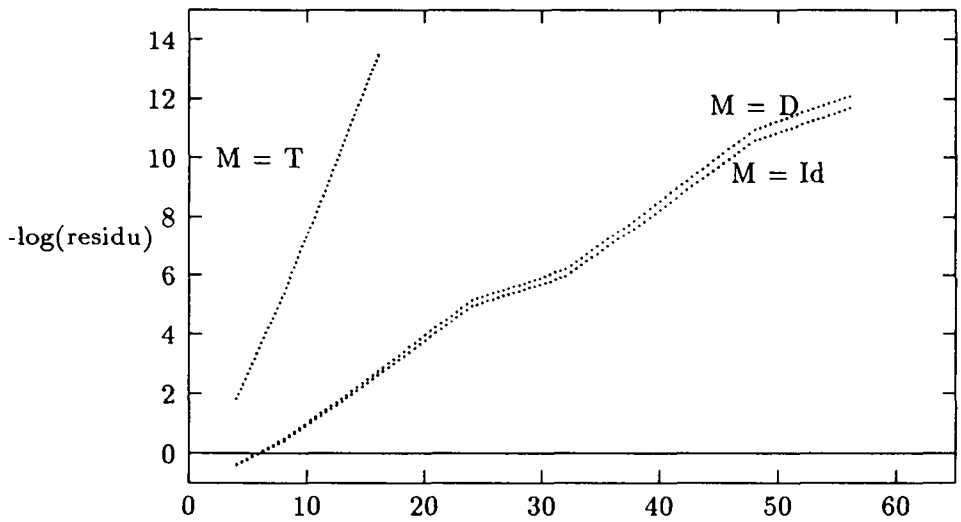

figure 2 (matrice 2)

Les tableaux 1 et 2 donnent les temps d'exécution sur quatre processeurs du cray2 pour ces deux matrices ainsi que l'accélération par rapport à un processeur; les deux préconditionnements $M = D$ et $M = T$ sont testés et le nombre de colonnes du second membre B prend les valeurs $k = 4$ et $k = 12$ (tableaux 1 et 2).

second membre:	Préconditionnement	
	$M = D$	$M = T$
$k = 4$ ($NBMAX = 24$)	nmult= 44 T=2.605s $\gamma = 3.66$	nmult= 44 T=2.913s $\gamma = 3.42$
$k = 12$ ($NBMAX = 96$)	nmult= 132 T=8.067s $\gamma = 3.70$	nmult= 132 T=8.285s $\gamma = 3.76$

Tableau 1. Temps d'exécution sur 4 processeurs et accélérations par rapport à un processeur : matrice 1 ($n = 2000, xp = 0.16, \epsilon = 10^{-10}$)

second membre:	Préconditionnement	
	$M = D$	$M = T$
$k = 4$ ($NBMAX = 24$)	nmult= 56 T=1.810s $\gamma = 3.16$	nmult= 16 T=0.851s $\gamma = 2.02$
$k = 12$ ($NBMAX = 96$)	nmult= 156 T=4.601s $\gamma = 3.67$	nmult= 48 T=1.492s $\gamma = 3.56$

Tableau 2. Temps d'exécution sur 4 processeurs et accélérations par rapport à un processeur : matrice 2 ($n = 1000, xp = 0.3, \epsilon = 10^{-10}$)

Pour ce type de problèmes nous obtenons de bonnes accélérations (supérieure à 3); la matrice 1 montre qu'à nombre d'itérations comparables le choix $M = D$ est meilleur que le choix $M = T$; par contre pour la matrice 2 le préconditionnement tridiagonal est plus adapté.

8.2 Exemple 3 (figures 3 et 4)

Nous étudions ici le comportement de l'algorithme dans le cas d'une matrice à diagonale dominante; nous utilisons une série de matrices creuses de même ordre engendrées aléatoirement et de densité donnée $xp = 2nz/n(n+1)$; les

éléments non-diagonaux, communs à toutes ces matrices, sont tirés dans l'intervalle $[-1/2, 1/2]$; les éléments diagonaux sont tirés dans $[0, diagscal]$ où diagscal est un coefficient que l'on fait varier.

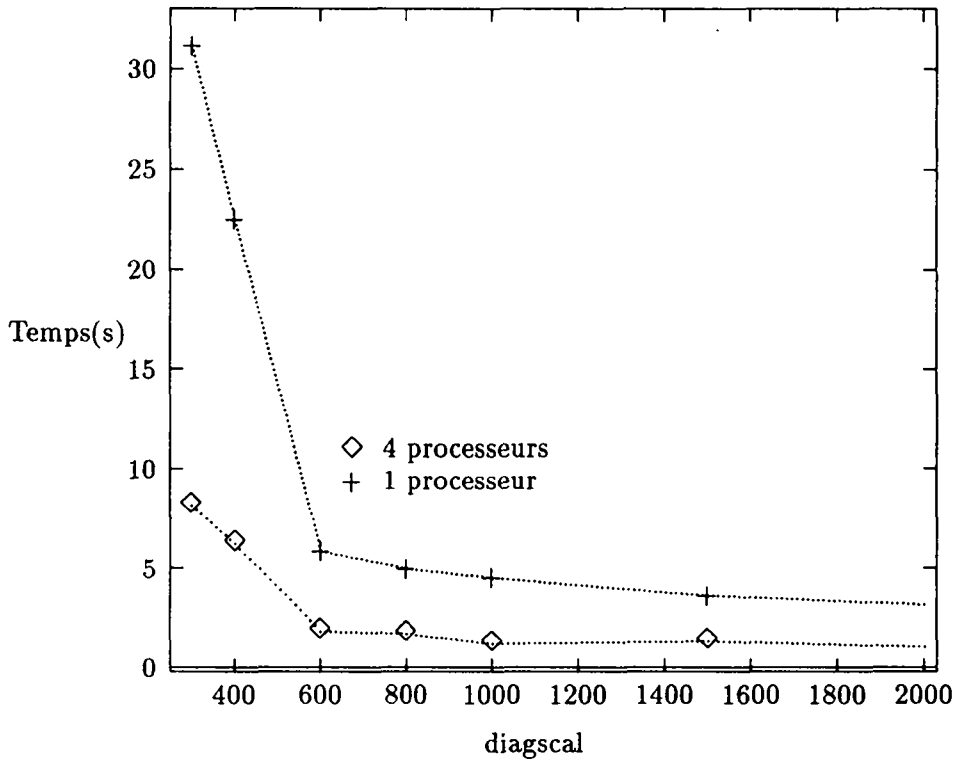

figure 3

Temps d'éxécution (ci-dessus) et accélérations (ci-dessous) quand la diagonale varie (éléments dans $[0, diagscal]$)
($n = 1000$, produit 2, $M = D$, $NBMAX = 50$, $\epsilon = 10^{-12}$,$xp = 2nz/n(n+1) = 0.15$,$nz = 75813$)

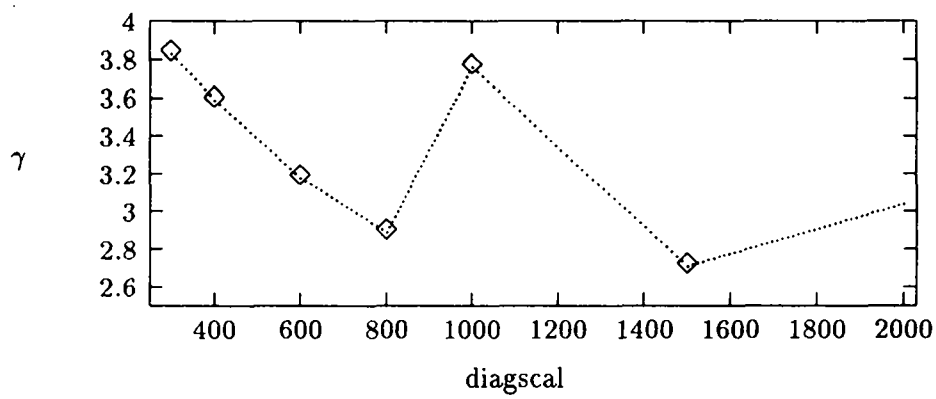

figure 4

Nous avons testé une série de telles matrices d'ordre $n = 1000$. Les tests sont réalisés avec un seul second membre; nous utilisons donc le produit 2 et nous préconditionnons par la diagonale; la figure 3 montre, comme on s'y attendait, que le temps d'exécution décroît quand le facteur *diagscal* augmente, c.a d. quand la dominance de la diagonale agmente; les accélérations obtenues sont cette fois encore bonnes (figure 4: entre 1.9 et 3.8).

8.3 Exemple 4 (tableau 3)

Nous nous interressons ici aux matrices issues des éléments finis par discrétisation sur une grille; considérons les matrices tridiagonales par blocs d'ordre n^2 suivantes :

$$A_n = \begin{pmatrix} T_1 & D_1 & & & \\ D_1 & T_2 & D_2 & & \\ & D_2 & T_3 & \ddots & \\ & & \ddots & \ddots & D_{n-1} \\ & & & D_{n-1} & T_n \end{pmatrix} \quad avec : \quad \begin{cases} D_i = -I_n \\ T_i = \begin{pmatrix} 4 & -1 & & \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{pmatrix} \in R^{n,n} \end{cases}$$

obtenues à partir du Laplacien sur une grille carrée. Nous utilisons les préconditionnements ILU et T; dans ce cas, le précontitionnement tridiagonal est parallélisé de manière naturelle puisque le système :

$$T * Y = X$$

peut s'écrire :

$$\begin{array}{l} do\ 10\ i = 1, n \\ \quad T_i * Y_i = X_i \\ 10\ continue \end{array}$$

si les données se mettent sous la forme :

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}, \; Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}$$

Tous les sous-systèmes $T_i * Y_i = X_i$ étant indépendants, la boucle {i} est exécutée en parallèle; nous notons cette deuxième version du préconditionnement tridiagonal : T'.

Nous prenons un exemple avec $n = 40$ noeuds soit une matrice d'ordre 1600; le tableau 3 résume les résultats obtenus (le produit bande utilisé est le produit 4 puisque l'on n'a qu'un second membre); le seuil utilisé pour la factorisation

incomplète $ILUTH$ est $\delta.||A_n||_1$ où δ est un seuil relatif que l'on a pris ici égal à 10^{-3} et le remplissage concerne le nombre $nz2$ d'éléments non nuls de la matrice, sans tenir compte de la symétrie.

	Préconditionnement : M			
	$ILU0$	$ILUTH$ (seuil relatif:10^{-3}) remplissage:7918→28406	T	T'
	nmult=54	nmult=14	nmult=178	
Temps (ms)	T=928ms $\gamma = 1.44$	T=234ms $\gamma = 1.36$	T=1931ms $\gamma = 1.65$	T=1351ms $\gamma = 2.37$

Tableau 3. Temps d'exécution sur 4 processeurs
et accélérations par rapport à un processeur :
matrice A_n avec $n = 40$, $n^2 = 1600$
$k = 1$, $NBMAX = 30$, $\epsilon = 10^{-10}$

On constate que les préconditionnements ILU donnent une vitesse de convergence nettement supérieure à celle obtenue avec le préconditionnement tridiagonal mais que, dans le cas présent d'un second membre unique, le parallélisme obtenu avec ces préconditionnements est faible; au contraire, le choix du préconditionnement tridiagonal parallélisé $M = T'$ conduit à un bien meilleur parallélisme.

8.4 Exemple 5 : Une matrice issue de la collection Harwell (figures 5 et 6)

Nous utilisons ici la matrice de la collection Harwell ([6]) que nous appelerons SMBPAM dont le titre est:

STIFNESS MATRIX BUCKLING PROBLEM (ANDY MERA)

Elle est d'ordre $n = 1224$ et a $nz = 28675$ éléments non nuls (partie triangulaire inférieure seulement); la figure 6 donne une image de cette matrice; notons que les éléments sont très regroupés autour de la diagonale principale (la 1/2-bande a pour largeur 56).
Cette matrice est mal conditionnée: $cond(A) \approx 7.10^9$ et les préconditionnements diagonal ou tridiagonal ne sont pas suffisants; nous choisissons donc pour M les préconditionnements ILU0 et ILUTH qui diminuent effectivement le conditionnement de manière significative bien que le remplissage peut être important suivant le seuil relatif choisi comme le montre le tableau suivant (nz2=56126

est le nombre d'éléments non nuls de la matrice et nz' est le nombre d'éléments non nuls dans la factorisation LU incomplète):

Préconditionnement : M			
$ILU0$	$ILUTH$ $seuil = 10^{-8}$	$ILUTH$ $seuil = 10^{-9}$	$ILUTH$ $seuil = 10^{-10}$
$cond(M^{-1}A) =$ 129.5	$cond(M^{-1}A) =$ 10257.7 $nz' = 65933$	$cond(M^{-1}A) =$ 433.8 $nz' = 84128$	$cond(M^{-1}A) =$ 11.6 $nz' = 95600$

Notons que cette matrice n'est pas positive, ce qui prouve que la méthode peut aussi s'appliquer dans ce cas, mais les critères de convergence sont plus difficiles à dégager!

Afin d'optimiser l'algorithme, nous avons ici ajouté au programme un contrôle d'efficacité à chaque itération; nous définissons ainsi l'efficacité eff_k à l'itération numéro k:

$$eff_k = 1/(\tau_k N_k)$$

où N_k est la complexité de l'itération k et τ_k le taux de croissance des résidus entre les itérations $k-1$ et k: $\tau_k = \frac{||R_k||}{||R_{k-1}||}$. Quand cette efficacité est jugée insuffisante on redémarre; le test adopté est:

$$eff_k \leq \overline{eff_k}(1-\alpha)$$

où $\overline{eff_k}$ est la moyenne des efficacités précédentes $eff_1, eff_2, ..., eff_k$; sur cet exemple, l'historique d'une exécution avec ou sans contrôle d'efficacité a montré que ce test (avec $\alpha = 0$) est satisfaisant au sens où le nombre d'itérations diminue: c'est à l'approche de la convergence que la méthode avec efficacité s'avère beaucoup plus rapide, alors qu'au début le contrôle est plutôt pénalisant (figure 5).

Remarque

Pour ne pas dépasser les limites de la mémoire, nous conservons la taille maximum de la base $NBMAX$ mais son influence sur le déroulement de l'algorithme devient négligeable: très rarement cette taille maximum est atteinte! On constate par contre que, sans contrôle d'efficacité, $NBMAX$ joue un rôle important: ainsi, quand ce paramètre est trop grand, on observe une stagnation périodique dans l'évolution des résidus, une accélération se produisant après chaque redémarrage quand la taille limite de la base est atteinte; par exemple pour la

matrice SMBPAM, il faut 135 iterations pour une precision $\epsilon = 3.e-13$ quand $NBMAX = 12$, 143 quand $NBMAX = 20$ et 386 quand $NBMAX = 40$..

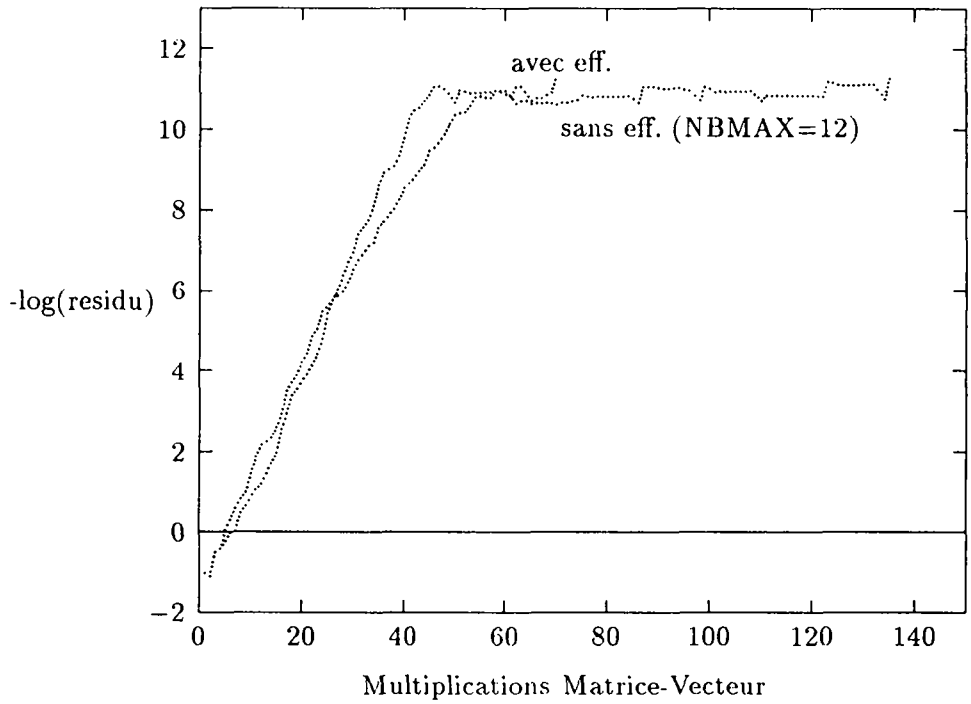

Figure 5: Matrice Harwell SMBPAM avec ou sans contrôle d'efficacité pour la précision relative $\epsilon = 3.10^{-13}$ (précision absolue $5.9795.10^{-12}$) n=1224, nz=28675, produit4, M=ILU0, k=1 second membre.

8.5 Exemple 6 : Influence des blocs sur la convergence(figure 6,tableau 4)

Nous avons d'autre part essayé, toujours sur l'exemple de la collection Harwell du paragraphe précédent, de voir si la méthode par bloc est avantageuse par rapport à la méthode simple (k=1) pour la résolution de plusieurs systèmes de même matrice, et d'autre part si la stratégie de parallélisation adoptée est optimale. Soit k le nombre de colonnes du second membre B que nous pouvons décomposer de plusieurs manières:

1. *un seul bloc*: c'est la méthode employée jusqu'ici
2. *par vecteurs*: $B = (b_1, b_2, ..., b_k)$, $b_i \in R^{n,1}$

3. *plusieurs blocs*: $B = (C_1, C_2, ..., C_{nproc})$, $C_i \in R^{n,(k/nproc)}$ où $nproc$ est le nombre de processeurs dont nous disposons;

Dans les deux derniers cas, la résolution de $AX = B$ où X se décompose en $X = (x_1, x_2, ..., x_k) = (X_1, X_2, ..., X_{nproc})$, est alors remplacée par la résolution en parallèle de systèmes linéaires indépendants:

$$\left[\begin{array}{l} doall\ i = 1, k \\ \quad x_i = A^{-1} b_i \end{array} \right.$$

ou:

$$\left[\begin{array}{l} doall\ j = 1, nproc \\ \quad X_j = A^{-1} C_j \end{array} \right.$$

pour lesquels nous employons la méthode de Davidson par blocs mais sur un seul processeur. Le tableau 4 rend compte d'une exécution de cet exemple sur le Cray2 en temps dédié avec $k = 16$ second membres. Nous utilisons le produit "bande" et le préconditionnement ILU0; pour la première méthode ces deux noyaux sont parallélisés comme nous l'avons fait jusqùà maintenant (produit 4), tandis que pour les deux autres méthodes ils sont exécutés séquentiellement; d'autre part, nous fixons le facteur $NBMAX$ dans chaque cas afin que l'espace de travail soit le même pour les trois stratégies (c'est environ: $2n(NBMAX)$, place nécessaire au stockage des vecteurs de la base V et de leurs produits par A, dans la méthode 1 et autant de copies de ces matrices que de processeurs dans les deux autres méthodes).

Méthode:	1 1 bloc de taille 16	2 4 blocs de taille 4	3 16 vecteurs
$NBMAX$:	128	32	32
Nmult :	224	268	435
Temps (s): 4 CPUS	2.637	2.460	3.396
Temps (s): 1 CPU	6,148	7.020	10.191
Accélération par rapport à un processeur:	2.3	2.9	3.0

Tableau 4: Exécution sur Cray2 de l'algorithme avec contrôle de l'efficacité et plusieurs stratégies de parallélisation pour la matrice Harwell SMBPAM (n=1224, nz=28675, $\epsilon = 10^{-10}$ (précision absolue:$2.079.10^{-9}$), produit 4, M=ILU0, k=16 second membres).

Nous constatons (tableau 4) que les deux dernières méthodes sont meilleures pour le parallélisme, mais que, sur cet exemple au moins, la convergence est plus lente quand la taille du bloc diminue: les valeurs du nombre total de multiplications nécessaires *nmult* vont en augmentant quand on passe d'une taille de bloc de 16 (méthode 1) à une taille de 4 (méthode 2), puis à une taille de 1 (méthode 3); on peut penser qu'un ....Un compromis entre parallélisme et vitesse de convergence nous amène donc à conseiller la deuxième stratégie qui a ici été la plus rapide. Cependant, il n'est pas évident que cette hiérarchie soit respectée dans tous les cas (pour d'autres matrices ou quand on ne s'impose plus de contraintes sur la place en mémoire...).
Dans les exemples suivants, nous n'utilisons plus de contrôle d'efficacité et nous reprenons la méthode habituelle (méthode 1).

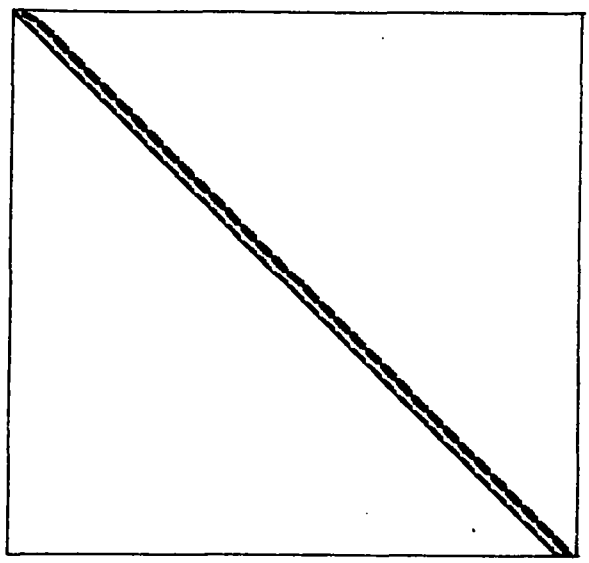

Figure 6: Image de la matrice Harwell SMBPAM

8.6 Importance des différentes étapes de l'algorithme (tableaux 5 et 6)

Pour terminer, donnons une idée des pourcentages du temps total d'exécution passés dans les phases importantes suivantes de l'algorithme (le temps total comprend le temps de préparation du préconditionnement plus le temps d'exécution de l'algorithme de Davidson lui-même) :

1. Produit matrice*bloc de vecteurs
2. Application du préconditionnement
3. Résolution des "petits" systèmes linéaires
4. Calcul des vecteurs $X_k = V_k Y_k$
5. Réorthogonalisation
6. Préparation du préconditionnement (factorisation ILU de A , factorisation LDL^t de T ,...)

Nous donnons deux exemples : un pour le produit bande (tableau 5) , l'autre pour le produit creux (tableau 6); le tableau 5 correspond à la matrice symétrique d'ordre n:

$$Tridiag(n,\alpha,\beta,D,ip) = \begin{pmatrix} d_1 & & & & & & & \\ \beta & d_2 & & & & & & \\ & \beta & d_3 & \ddots & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & & & \\ \alpha & & & & \ddots & \ddots & & \\ & \ddots & & & & \ddots & \ddots & \\ & & \alpha & & & & \beta & d_n \end{pmatrix}$$

$$ligne\ ip+1 \rightarrow$$

avec les paramètres suivants: $n = 200$, $D = 2I_n$, $\alpha = -0.1$, $\beta = -1$, $ip = n - 1$.

Son conditionnement est de $1.6.10^4$; le produit bande utilisé est le produit 4 et on prend: $k = 4$, $NBMAX = 20$, $\epsilon = 10^{-8}$.

On remarque que le coût du produit est ici négligeable par rapport à celui du préconditionnement .

Le tableau 6 correspond à la matrice à diagonale dominante construite comme il est expliqué dans le paragraphe §8.1 pour la matrice 1, mais avec les paramètres: $n = 200$, $, xp = 0.2$, $diagscal = 20$, $subscal = 0.1$;

le produit creux utilisé est le produit 1 et on prend: $k = 4$, $NBMAX = 24$, $\epsilon = 10^{-10}$.

Cette fois le produit s'avère très couteux; d'autre part on remarque l'écart important de coût entre préconditionnement diagonal, pratiquement gratuit, et préconditionnement tridiagonal .

phases	$M = ILU0$	$M = ILUTH$	$M = T$
1(produit)	0.84	0.80	1.21
2(prec.)	43.67	41.89	21.13
3(resol.sys.)	2.9	2.4	3.3
4(calcul X_k)	43	38.92	36.8
5(ortho.)	4.2	4	5.4
6(prepar.prec.)	0.8	4.6	4

Tableau 5. Pourcentages de temps dans chaque partie de l'algorithme (produit bande)

phases	$M = D$	$M = T$
1(produit)	62.54	53.28
2(prec.)	0.12	14.82
3(resol.sys.)	3.7	1.8
4(calcul X_k)	27.2	20.1
5(ortho.)	3.4	2.3
6(prepar.prec.)	0.002	1.7

Tableau 6. Pourcentages de temps dans chaque partie de l'algorithme (produit creux)

9 Conclusion

Nous avons montré dans cet article que la méthode de Davidson peut s'appliquer à la résolution de systèmes linéaires de grande taille et que son implémentation par blocs est efficace sur un multiprocesseur. Bien que la convergence n'a été démontrée que dans le cas particulier où la matrice est *symétrique définie positive*, cette méthode peut s'appliquer avec succès à certaines matrices qui ne sont pas positives. Outre une meilleure efficacité dans les calculs, l'utilisation des blocs pour résoudre simultanément des systèmes de même matrice peut accélérer la convergence; cet effet est sans doute étroitement lié à la distribution

spectrale de la matrice et la taille de bloc choisie. Un prolongement intéressant de cette étude serait donc de regarder si ce phénomène est commun aux méthodes de sous-espaces utilisant un espace de Krylov comme les gradients conjugués ou GMRES ([2] ,[11]).

Bibliographie

[1] M. Arioli, J. W. Demmel, I. S. Duff. *Solving sparse linear systems with sparse backward error.* SIAM J. Matrix Anal. Appl.,Vol 10, No. 2, pp 165-190, April 1989.

[2] S. F. Ashby, T. A. Manteufel, P. E. Saylor. *A Taxonomy for Conjugate Gradient Methods* Report No. UIUCDS-R-88-1414, Department of Computer Science, University of Illinois, Urbana-Champain, March 1988.

[3] F. L. Bauer. *A further generalization of the Kantorovic inequality.* Numerische Mathematik 3, 117-119. 1961.

[4] M. Crouzeix, M. Sadkane. *Sur la convergence de la méthode de Davidson.* C. R. Acad. Sci. Paris, t. 308, Série I, 1989.

[5] E. R. Davidson *The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices.* J. Comp. Phys., 17, 1975, p.87-94

[6] I.S. Duff, R.G. Grimes,J.G. Lewis. *Sparse Matrix Test Problems* Report CSS 191, Harwell Laboratory,England, 1987.

[7] J. Erhel. *Sparse Matrix Multiplication on Vector Computers* Rapport de Recherche INRIA numéro 1101, Octobre 1989.

[8] G.H. Golub,C.F. Van Loan. *Matrix computations* Second edition, Johns Hopkins University Press, 1989.

[9] B. N. Parlett. *The symmetric eigenvalue problem.* Englewood Cliff, N.J.,1980

[10] B. Philippe, Y. Saad, W. J. Stewart. *Numerical Methods in Markov Chain Modeling.* Publication interne IRISA/INRIA numéro 495, Septembre 1989.

[11] Y. Saad, M. Schultz. *GMRES: A Generalized Minimal Residual Algorithm for solving nonsymmetric linear systems.* SIAM J. Sci. and Stat. Comp. 7, pp 856-869, 1986.

[12] M. Sadkane. *Analyse numérique de la méthode de Davidson.* Eléments finis C^1 de degré quatre pour une triangulation équilatérale. Thèse de l'université de Rennes I. 1989

LISTE DES DERNIERES PUBLICATIONS INTERNES PARUES A L'IRISA

PI 525 **MULTI-SCALE AUTOREGRESSIVE PROCESSES**
Michèle BASSEVILLE, Albert BENVENISTE
Mars 1990, 136 Pages.

PI 526 **TRANSFORMATIONS PYRAMIDALES D'IMAGES NUMERIQUES**
Nadia BAAZIZ, Claude LABIT
Mars 1990, 100 Pages.

PI 527 **LE LANGAGE SIGNAL : UN EXEMPLE EN SEGMENTATION AUTOMATIQUE DE LA PAROLE CONTINUE**
Claude LE MAIRE
Mars 1990, 112 Pages.

PI 528 **CONDITIONAL REWRITE RULES AS AN ALGEBRAIC SEMANTICS OF PROCESSES**
Eric BADOUEL
Mars 1990, 46 Pages.

PI 529 **RESEAUX SYSTOLIQUES SPECIFIQUES A BASE DU PROCESSEUR API15C**
Patrice FRISON, Eric GAUTRIN, Dominique LAVENIER, Jean-Luc SCHARBARG
Mars 1990, 26 Pages.

PI 530 **SEMI-GRANULES AND SCHIELDING FOR OFF-LINE SCHEDULING**
Bernard LE GOFF, Paul LE GUERNIC, Julian ARAOZ DURAND
Avril 1990, 46 Pages.

PI 531 **DATA-FLOW TO VON NEUMANN : THE SIGNAL APPROACH**
Paul LE GUERNIC, Thierry GAUTIER
Avril 1990, 22 Pages.

PI 532 **OPERATIONAL SEMANTICS OF A DISTRIBUTED OBJECT-ORIENTED LANGUAGE AND ITS Z FORMAL SPECIFICATION**
Marc BENVENISTE
Avril 1990, 100 Pages.

PI 533 **ADAPTATION DE LA METHODE DE DAVIDSON A LA RESOLUTION DE SYSTEMES LINEAIRES : IMPLEMENTATION D'UNE VERSION PAR BLOCS SUR UN MULTIPROCESSEUR**
Miloud SADKANE, Brigitte VITAL
Avril 1990, 34 Pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399